Scuola universitaria professionale della Svizzera italiana
Dipartimento tecnologie innovative

SUPSI

Viki: Smart Home Natural Language interface

Relatore

Nicola Rizzo

Studente/responseCode

Luca Ambrosini

Correlatore

Alan Ferrari

Committente

-

Corso di laurea

-

Modulo

M00002 Progetto di diploma

Anno

2015/16

Data

17 agosto 2016

Capitolo 1

Introduzione

Disegnare una macchina in grado di comportarsi come un umano, in particolare di parlare e interpretare il linguaggio, è uno degli obbiettivi dell'ingegneria sin da metà del 20esimo secolo. Le interfacce in linguaggio naturale sono considerate come il punto di arrivo dell'interazione uomo macchina. Lo sviluppo in questo campo è stato molto intenso negli ultimi anni, ciò ha permesso la realizzazione di agenti intelligenti, che simulino una conversazione con la persona e che riescano a compiere azioni più complesse di semplici comandi con frasi standardizzate.

1.1 Cenni storici

Il primo esempio nella storia di dispositivo ad interazione vocale è da collocare nell'estate del 1952, presso i laboratori Bell. Quell'anno vennero eseguiti i primi test di "Audrei" (Automatic Digit Recognizer), un dispositivo in grado di comporre un numero di telefono dettato ad un microfono.
Nel 1962 IMB presentò "Shoebox", una macchina in grado di comprendere 16 diverse parole pronunciate in inglese. Questa macchina era destinata ad essere una calcolatrice vocale.
Lo sviluppo di sistemi in grado di comprendere il linguaggio naturale è poi proseguito nel tempo, passando dalla comprensione di pochi suoni alla comprensione continua del linguaggio naturale; le tecniche si sono evolute passando da metodi statistici fino ad approcci basati sul deep learning. Esso è una branchia del machine learning, che simula delle reti neurali multi strato che riescono ad apprendere funzioni complesse. [**?**]
Grossi miglioramenti in questo campo sono pervenuti nell'ultimo secolo, soprattuto grazie all'incremento delle capacità computazionali. Questo ha permesso la realizzazione di agenti intelligenti sempre più complessi.

1.2 Evoluzione degli agenti

I primi dispositivi ad interazione vocali sono gli "Interactive Voice Response", cioè gli agenti dei call center, che descrivono attraverso la voce i comandi e ricevono input attraverso i numeri digitati sul telefono. Il numero di input era quindi molto ridotto e la struttura della conversazione era fissa.
Successivamente i lettori automatici e i dispositivi ad interazione vocale sono stati integrati nei sistemi operativi. La loro funzione principale consisteva nell'aiutare le persone con delle disabilità. Era comunque necessario un microfono, quindi una prossimità al computer. Inoltre la voce aveva una funzione di sostituzione delle capacità visive o motorie; non erano previste funzionalità dedicate che permettessero una maggior produttività.
Con gli smartphone, che sono dotati di un microfono e che dispongono di una connessione a Internet, gli agenti intelligenti sono diventati parte della nostra vita quotidiana. Vista la limitata capacità di calcolo degli smartphone tutto il processamento dell'informazione viene eseguito attraverso cloud computing, che utilizza tecniche di deep learning. [**?**]
L'ultima generazione di dispositivi ad interazione vocale è costituita da "Amazon Echo" e "Microsoft Kinect", essi sono in grado di ricevere input vocali in modo continuo, senza che l'utente debba avere un microfono appresso e senza che sia necessario azionare un dispositivo. Questo ha portato l'interazione vocale ad un nuovo livello di usabilità e ha aperto nuove possibilità di utilizzo di questa tecnologia, in particolare nell'ambito delle smart home.

1.3 Il momento giusto

Storicamente lo scetticismo a proposito delle interfacce in linguaggio naturale è sempre stato molto elevato: soprattuto per la loro scarsa produttività sono sempre state considerate un accessorio e non una tecnologia che potesse essere sfruttata. Ora però tutte le tecnologie necessarie alla realizzazione di un agente intelligente che ci possa aiutare nella vita quotidiana sono pronte:

- **Speech To Text:** Grazie alle tecniche di machine learning sviluppare negli ultimi anni, questa tecnologia è arrivata ad alti livelli di accuratezza, superando in alcuni casi perfino le capacità di percezione dell'uomo. Sono ormai disponibili componenti che eseguono speech-to-text in tutte le lingue del mondo.[**?**]

- **Comprensione del testo:** L'analisi semantica, la vettorizzazione di parole e frasi, permettono una sempre maggior strutturazione del contenuto

del testo, la quale consente una migliore comprensione da parte delle macchine.[**?**]

- **Connessione:** La capacità di calcolo richiesta per effettuare STT e comprendere un testo è molto elevata, per questo in genere si ricorre a un server remoto; l'incremento della larghezza di banda e la diminuzione dei tempi di latenza hanno reso possibile delle risposte in tempi adeguati.
- **Audio always on:** La tecnologia ha permesso la creazione di dispositivi che ascoltano in modo continuo e sono in grado di riconoscere delle keyword per la loro attivazione ("Ehi Siri"), le persone inoltre si sono abituate e hanno imparato ad accettare questa profonda invasione della privacy.
- **IOT:** Si stima che il mercato dell'IOT raggiunga una cifra d'affari di 1200 Miliardi di $ entro il 2020 e l'home automation è uno dei settori nei quali un agente può raggiungere la sua massima utilità.

Capitolo 2

Caso d'uso

L'utilità degli Agenti Intelligenti ad interazione vocale è spesso messa in dubbio, ma ci sono alcune occasioni nelle quali le loro capacità brillano, poiché forniscono un'esperienza d'uso diversa dalle interfacce basate su schermi o touch:

- **Accessibilità**: Consentono un'esperienza d'uso soddisfacente a persone con disabilità motorie o visive, in quanto la procedura di descrizione delle operazione possibili e la successiva richiesta di un input non è più necessaria, gli agenti possono infatti eseguire comandi in risposta a frasi come "Manda un messaggio a Mario dicendo che arriverò tardi"
- **Eye-busy o Hand-busy**: In scenari quali la guida o attività svolte in cucina, in cui si hanno le mani impegnate e non si ha la possibilità di concentrare la propria attenzione su uno schermo, gli agenti Intelligenti diventano particolarmente utili.
- **Automazione casalinga**: Supportare la creazione di comandi complessi a discrezione dell'utente, che permettano di compiere azioni in modo semplificato. Ad esempio "Buonanotte" potrebbe automaticamente abbassare tutte le tapparelle e spegnere tutte le luci.

In queste situazioni sarebbe quindi ideale avere un agente in grado di svolgere per noi la maggior parte delle operazioni che gli vengono indicate attraverso la voce, come se stessimo conversando con una persona alla quale chiediamo di svolgere il compito.

Capitolo 3

Obbiettivo

Il progetto è basato su "Viki", un Agente Intelligente, capace di controllare molti degli apparecchi presenti in un abitazione e di fornire informazioni, ad esempio riguardanti il meteo.[**?**] Esso è stato sviluppato presso l'Istituto Sistemi Informativi e Networking. [**?**] Il primo obbiettivo del progetto di bachelor consiste nella comprensione dell'infrastruttura del sistema attuale; successivamente si vuole migliorare l'interazione vocale con il sistema, cercando di renderla il meno rigida possibile. Inoltre si implementeranno strutture atte a migliorare l'intelligenza dell'agente.

3.1 Interfaccia in linguaggio naturale

3.1.1 Grammatiche fisse

Il sistema attuale prevede l'interazione vocale, ma utilizza un sistema basato su delle grammatiche fisse. Questo implica quindi una struttura della frase definita a priori dal programmatore, che nel caso non sia rispettata, impedisce la comprensione del comando da parte dell'agente.

3.1.2 Rimozione dei vincoli

Il progetto aspira a creare un interfaccia libera da questi vincoli, che provi a comprendere il senso della frase in modo indipendente dai singoli vocaboli e dalla struttura utilizzata. Grazie all'interfaccia libera l'utilizzatore può concentrarsi sull'azione da eseguire e meno su come esprimerla per far si che l'agente sia in grado di comprenderla. Una delle critiche che viene più spesso mossa alle interfacce in linguaggio naturale è la necessità dell'utilizzatore di compiere uno sforzo mentale per pensare come la macchina. Grazie alla rimozione di questi vincoli l'utilizzatore dovrebbe trovare l'interazione con l'agente più simile

a una conversazione tra persone, garantendo quindi una maggior soddisfazione. Un'interfaccia di questo livello semplificherebbe l'utilizzo di una smart home al punto di renderla fruibile anche a persone che non si trovano normalmente a loro agio con la tecnologia.

3.2 Incremento delle API

3.2.1 API attuali

Le capacità del sistema sono strettamente collegate alla mole di informazioni alle quali esso ha acceso e ai dispositivi che è in grado di controllare. Al momento Viki può controllare :

- Lampadine philips HUE (accensione, colorazione, intensità)
- Prese di corrente z-wave (accensione, lettura potenza istantanea)

e ha accesso alle seguenti informazioni:

- Sensori di movimento, luminosità, umidità, temperatura
- Previsioni meteo (yahoo)

3.2.2 API future

Durante lo sviluppo del progetto di bachelor si vogliono incrementare le capacità del sistema, in particolare Viki dovrà essere in grado di controllare:

- Tapparelle motorizzate
- Mediacenter
- Impostazione di sveglie
- Impostazione di promemoria
- Aggiunta eventi calendario
- Impostazioni timer

e avrà accesso a informazioni aggiuntive quali :

- Palinsesto televisivo (RSI, Mediaset, Rai)

Capitolo 4

Stato dell’arte

Capitolo 5

Architettura

Il sistema sviluppato durante il progetto di tesi si compone di diverse componenti :

- **Orecchie**: Riconosce la keyword di inizio conversazione, trasforma in testo quanto detto dall'utente. (Python)
- **Cervello**: Trasforma il testo ricevuto dalle orecchie in un comando, che viene comunicato al "Braccio". Si occupa inoltre di modificare lo stato della conversazione di orecchio e voce. (Java)
- **GUI**: Alternativa a orecchie e voce, sviluppata principalmente per motivi di debug. (NodeJs + Electron)
- **Voce**: Notifica all'utente quando un comando viene eseguito, quando non viene compreso e gli pone domande se necessario. (Python)
- **Braccio**: Comunica al cervello quali comandi offre e riceve i comandi da eseguire. (NodeJs)

5.1 Vantaggi

Si è scelto di separare il sistema in diverse componenti per favorirne la riutilizzabilità. Si è cercato di ridurre al minimo le interdipendenze, cercando di rendere ogni componente autonomo e riutilizzabile in altri contesti.
Motivazione principale per la separazione in componenti è stata la maggior libertà tecnologica, infatti i vari moduli sono realizzati in linguaggi diversi. Questa scelta ha consentito utilizzo del miglior linguaggio, con le migliori libreria, per ogni componente del sistema.

5.2 Svantaggi

La separazione in moduli ha però richiesto uno sforzo superiore alla realizzazione di un singolo componente, in quanto è stato necessario creare degli appositi canali di comunicazione tra i vari moduli; si è inoltre resa necessaria la definizione di formalismi per la trasmissione dell'informazione.
Infine la trasmissione di informazione su un canale è più onerosa e lenta rispetto alla condivisione dello stesso spazio di memoria, ma per la quantità di informazione che è necessario trasmettere questa differenza non è stata ritenuta significativa.

Capitolo 6

Comunicazione tra componenti

I diversi canali di comunicazione hanno esigenze diverse, abbiamo quindi utilizzato per tecnologie diverse a seconda del canale.

6.1 Braccio-Cevello

Il cervello deve essere informato dal braccio a proposito di quali sono i comandi che possono essere eseguiti, quest'operazione avviene una sola volta, durante la fase di inizializzazione del sistema.
L'informazione viene reperita dal braccio, seguendo l'architettura REST, attraverso una chiamata all'indirizzo GET *"/metadata"*, all'indirizzo del braccio. Essa ritorna una struttura dati JSON che descrive le capacità il sistema, seguendo il formalismo documentato nel capitolo apposito.

6.2 Altre comunicazioni

I canali di comunicazione orecchie-cervello e cervello-braccio vengono utilizzati in ogni interazione con l'utente, per questo motivo si è deciso di non seguire l'architettura REST, ma di utilizzare dei canali di comunicazione di tipo socket.
Per la realizzazione di questo tipo di canale è stata scelta la tecnologia socket.io.

6.2.1 Socket.io

Socket.io si occupa di creare un canale di trasmissione che una comunicazione bi-direzionale; essa si basa sull'emissione di eventi, che possono essere divisi in categorie attraverso l'utilizzo dei namespace.
Questa tecnologia garantisce un livello di astrazione dalla tecnologia di trasporto utilizzato. Inoltre si occupa di gestire autonomamente il failover : quando

il client o il server non sono disponibili i messaggi vengono messi in coda e verranno inviati alla riconnessione del dispositivo.
Questa tecnologia è stata scelta anche per la disponiblità di implementazioni in tutti i linguaggi utilizzati:

- **Python client**: https://pypi.python.org/pypi/socketIO-client
- **Python server**: https://github.com/miguelgrinberg/python-socketio
- **Nodejs client**: https://www.npmjs.com/package/socket.io-client
- **NodeJs server**: https://www.npmjs.com/package/socket.io
- **Java client**: https://github.com/socketio/socket.io-client-java
- **Java server**:https://github.com/mrniko/netty-socketio

Capitolo 7

Comunicazione X - Cervello

Il cervello possiede una struttura che permette l'utilizzo di canali di comunicazione input. Nell'implementazione attuale i comandi vengono ricevuti attraverso un socket server esposto sulla porta 8887.

7.1 Input

Su questo canale possono pervenire messaggi qualificati dal namespace "textCommand", l'oggetto trasmesso nel messaggio deve essere una stringa e corrisponde al testo che contiene il testo pronunciato dall'utente e dal quale il cervello deve provare a estrarre un comando.

7.2 Output

Non appena il riconoscimento del comando è stato completato, il cervello manda una risposta attraverso l'ack di risposta al messaggio ricevuto.
La risposta rappresenta lo stato del comando che è stato ricevuto; le possibile risposte sono :

- **UNKNOWN**: Qualcosa di inaspettato è successo durante l'elaborazione, probabilmente un errore interno del server.
- **OK**: E' stato possibile trovare un comando valido nel testo, verrà quindi inviato attraverso i canali di comunicazione selezionati.
- **LOW_CONFIDENCE**: Il sistema non è riuscito a identificare un comando con un grado di sicurezza sufficiente a procedere con la sua esecuzione.
- **MISSING_NUMBER**: E' stato trovato un riferimento a un comando, che ha un parametro obbligatorio di tipo NUMBER mancante.

- **MISSING_COLOR**: E' stato trovato un riferimento a un comando, che ha un parametro obbligatorio di tipo COLOR mancante.
- **MISSING_DATETIME**: E' stato trovato un riferimento a un comando, che ha un parametro obbligatorio di tipo DATETIME mancante.
- **MISSING_FREE_TEXT**: E' stato trovato un riferimento a un comando, che ha un parametro obbligatorio di tipo FREE_TEXT mancante.
- **LEARNED**: Il nuovo comando è stato appreso con successo.
- **TEACH**: Il sistema ha riconosciuto il desiderio di insegnamento ed è pronto ad apprendere.

Capitolo 8

Comunicazione Braccio - Cervello

Il modulo di interazione determinazione del comando è stato realizzato come un componente esterno dal sistema di gestione dell'abitazione, cioè quello che si occupa di accedere alle informazioni e di azionare gli attuatori (braccio). E' stato quindi necessario definire un protocollo che informasse il sistema di controllo vocale di quali operazioni possono essere compiute e quali tipologie di informazioni sono disponibili.

8.1 Struttura dell'informazione

Per definizione l'insieme delle operazioni che il sistema di gestione è in grado di compiere abbiamo definito la seguente struttura:

- **Universe**: l'insieme di tutti i domini.
- **Domain**: un oggetto o un dominio di informazione che il sistema rende disponibile, per essere azionata o interrogata (es. Lampada, Tapparella, Meteo, Palinsesto)
- **Operation**: sono definite nell'ambito di un dominio e rappresentano le operazioni che possono essere richieste (es accensione di una luce, richiesta delle previsioni metereologiche)
- **Parameters**: sono definiti nell'ambito di un operazioni e rappresentano i parametri che possono essere associati a un operazione (es. colore da impostare per la lampada, luogo per le previsioni metereologiche)
- **ParameterType**: i parametri precedentemente definiti devono essere di una tipologia specifica(es. Data, Luogo)

8.2 Formalismo

Per la comunicazione della struttura precedentemente definita tra l'agente intelligente e l'interfaccia vocale si è scelto di utilizzare il formato JSON

8.2.1 Universe

Nome	Descrizione	Tipo
id	Identificativo univoco	String
domains	Lista dei domini che compongono l'universo	JSONArray di Domain

Tabella 8.1: Struttura JSON Universe

8.2.2 Domain

Nome	Descrizione	Tipo
id	Identificativo univoco	String
words	Parole associate al dominio (es. light,lamp)	JSONArray di String
friendlyNames	Nomi associate al dominio (es. "palla" -> lampada)	JSONArray di String
operations	Operazioni che possono essere eseguite nel dominio	JSONArray di Operation

Tabella 8.2: Struttura JSON Domain

8.2.3 Operation

Nome	Descrizione	Tipo
id	Identificativo univoco	String
words	Parole associate al dominio (es. light,lamp)	JSONArray di String
textInvocation	Frasi per invocare l'operazione	JSONArray di String
mandatoryParameters	Parametri obbligatori per l'operazione	JSONArray di Parameter
optionalParameters	Parametri opzionali, non necessari	JSONArray di Parameter

Tabella 8.3: Struttura JSON Operation

8.2.4 Parameter

Nome	Descrizione	Tipo
id	Identificativo univoco	String
missingParams	Tipo del parametro	ParameterType

Tabella 8.4: Struttura JSON Parameter

8.2.5 Tipologie di parametri

Il sistema supporta parametri tipizzati, che possono appartenere alle seguenti categorie:

- LOCATION
- DATETIME
- NUMBER
- COLOR
- FREE_TEXT

Capitolo 9

Comunicazione Cervello - Engine

Il software di gestione vocale si occupa di estrarre i comandi che l'utente ha richiesto al sistema. Dopo aver completato il processamento dell'informazione invia la serializzazione in formato JSON di un oggetto di tipo Command.

9.1 Command

L'oggetto restituito rappresenta il comando che deve essere eseguito dal sistema, include inoltre la frase che l'utente ha pronunciato e la frase che nel sistema è associata al comando riconosciuto.

9.1.1 Struttura JSON Command

Nome	Descrizione	Tipo
confidence	probabilità di correttezza del comando	Double
said	Frase ascoltata	String
domain	Id del dominio	String
operation	Id dell'operazione	String
status	Stato del comando ricevuto	CommandStatus
understood	Frase associata al comando nel sistema	String
parameters	Lista di parametri e relativi valori	JSONArray di ParamValue

Tabella 9.1: Struttura JSON Command

9.1.2 Struttura JSON ParamValue

Nome	Descrizione	Tipo
id	Id del parametro	String
type	Tipologia del parametro	ParamType
value	Valore assunto dal parametro	String

Tabella 9.2: Struttura JSON ParamValue

Capitolo 10

Orecchie

$$Voice \implies \underbrace{Jarvis,}_{\text{Hot Keyworld}} \underbrace{could\ you\ please\ create\ a\ romantic\ atmoshpere?}_{\text{Speech to text}}$$

I sistemi di riconoscimento vocale sono quasi nella totalità dei casi basati su un sistema di regole. Il sistema di grammatiche viene quindi utilizzato sia per il riconoscimento di quanto detto dall'utente, sia per il riconoscimento del comando.
Lo scopo del progetto consiste nella rimozione di queste grammatiche, è quindi stato necessario realizzare un nuovo componente che fosse in grado di trasformare in testo i comandi espressi vocalmente dell'utente.

10.1 Problemi

Il sistema che si desidera realizzare deve essere in grado di riconoscere parlato in modo continuo, senza che l'utente debba azionare manualmente la conversazione. Deve inoltre essere in grado di comprendere qualsiasi tipo di testo, senza conoscere a priori nessuna informazione a proposito delle parole che verranno utilizzare o della sintassi della frase.

- **Offline**: Al momento non esistono dei prodotti non commerciali che riescono ad effettuare trascrizioni di testo in modo continuo senza un accesso a internet. Nei moderni sistemi operativi, come ad esempio in MAC OSX Yosemite, è presente un sistema di attivazione vocale di una dettatura, ma essa non viene interrotta automaticamente dopo aver pronunciato una frase. Abbiamo inoltre ritenuto opportuno non sviluppare un sistema che si basasse su un API offerta da uno specifico sistema operativo.

L'alternativa oper-source è costituita da CMU-SPHINX, questo software è in grado di riconoscere una keyword di attivazione e dopo di essa del testo, ma esso deve seguire delle grammatiche precise, non è quindi utilizzabile nel nostro prodotto.

- **Online**: Al momento uno dei migliori software di trascrizione del testo è offerto da Google, attraverso un sistema di API; esse offrono la trascrizione di 80 diverse lingue, con un ottima affidabilità. Le API Google però non permettono il riconoscimento in modo continuo, ma solo di una breve sequenza di testo che rappresenta la frase. Non è quindi possibile utilizzarle anche per il riconoscimento della parole di attivazione.

Vista la mancanza di un prodotto con tutte le caratteristiche richieste si è reso necessario realizzare un modulo, che utilizzasse le caratteristiche peculiari di varie API, per ottenere quanto desiderato.

10.2 Soluzione

Si è scelto di realizzare un sistema che riconoscesse, senza utilizzare una connessione a un API, una parola di attivazione; essa porta quindi all'attivazione del modulo di riconoscimento attraverso l'API offline. Grazie a questo sistema combinato l'utilizzo delle API viene ridotto al minimo indispensabile, mantenendo al minimo l'utilizzo della rete e minimizzando eventuali costi; questo inoltre garantisce la miglior tecnologia di trascrizione e permette di poter cambiare lingua in maniera molto semplice.

10.3 Scelta tecnologica

Il componente di riconoscimento vocale è realizzato come un modulo indipendente, si è quindi potuto cambiare linguaggio di programmazione. Si è scelto di utilizzare python, per la presenza di due librerie di riconoscimento di una parola di attivazione (Pocket-Sphinx e SnowBoy) e per la presenza di un'ottima libreria di STT che garantisce un livello di astrazione superiore alle API dei singoli produttori.

10.4 Hot Keyword detection

Per il riconoscimento di una parola che inizii l'interazione vocale sono state valutate principalmente due alternative :

- **SnowBoy**: Permette il riconoscimento di sequenze di testo multiple, non comprende un motore di trascrizione del testo [**?**]
- **Online**: Utilizza il motore del noto prodotto opensource CMU-SPHINX per riconoscere una parola, che lancia il normale motore di trascrizione del testo.[**?**]

Si è scelto di utilizzare la prima alternativa, in quanto il riconoscimento viene effettuato tramite il confronto con un modello che può essere allenato da più persone, migliorando l'affidabilità del sistema. SnowBoy inoltre riesce a riconoscere una sequenza di testo utilizzando delle risorse davvero esigue, i produttori dichiarano un utilizzo di RAM minore di 20MB e un utilizzo di CPU inferiore al 10%, sul primo modello di raspberry-py. [**?**]

10.5 Speech To Text

Dopo aver riconosciuto la hot keyword viene avviato il sistema di riconoscimento vocale. Questo viene fatto attraverso la libreria python SpeechRecognition. Essa fornisce un livello di astrazione sopra le principali API di speech to text (Google Speech Recognition, Wit.ai, Microsoft Bing Voice Recognition, api.ai, IBM Speech to Text).[**?**]
Il sistema riconosce la frase ,delimitata da un breve silenzio, e la invia alle API sopra citate; nel caso nessuna di essa dovesse essere disponibile viene trascritta utilizzando CMU Sphinx, il quale funziona offline, ma la qualità della trascrizione non è paragonabile a quanto viene effettuato dalle API.
Una volta che la sequenza è stata riconosciuta essa viene inviata, attraverso il canale socket.io, al cervello per la sua elaborazione.

Capitolo 11

Voce

Il software prevede un interazione completamente vocale, si è quindi reso necessario comunicare con l'utente attraverso la voce. L'utente riceve quindi un messaggio dopo aver inviato un comando, che lo informa a proposito della riuscita del comando o a riguardo l'errore che è stato generato. Maggiori dettagli sono presenti nella sezione dedicata alla struttura della conversazione.

11.1 Text To Speech

Il modulo di text to speech è realizzato insieme al modulo di speech to text, anch'esso è quindi realizzato in python. Per semplicità si è scelto di delegare al sistema operativo il compito di effettuare lo speech to text. Nell'implementazione attuale si è scelto di utilizzare il sistema operativo MAC OSX, quindi si è utilizzato il comando di sistema *say*.

11.2 Risposte

Quando il software riceve un messaggio di risposta dopo l'esecuzione di un comando viene scelta casualmente una frase, coerente con lo stato attuale del sistema. Allo stato attuale non è presente un algoritmo di generazione automatica della frase che viene pronunciata.

11.3 Struttura conversazione

Ogni conversazione deve iniziare con la hot keyword. Dopo di essa viene riconosciuto un comando.
Nel caso nella frase pronunciata venga trovato un comando, il sistema lo esegue, informa l'utente e torna allo stato iniziale. Altrimenti informa l'utente a

proposito dell'errore che è avvenuto e si prepara a ricevere un'altro comando, che non è necessario inizi con la hot keyword.

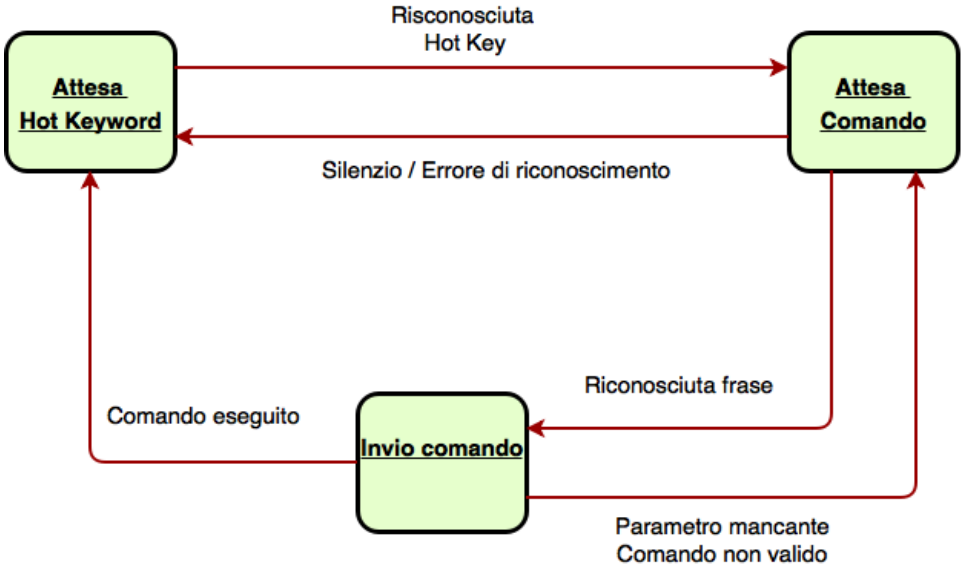

Figura 11.1: Diagramma di stato di una conversazione

Capitolo 12

GUI

Si è scelto di sviluppare un'interfaccia grafica che permettesse di interagire con il sistema senza passare dall'interazione vocale.
Essa si è resa necessaria per scopi di debug del codice, ma può essere utile quando non si può parlare o quando si vogliono avere maggiori informazioni a proposito di cosa sta accadendo nel sistema.

12.1 Scelta tecnologica

Inizialmente l'interfaccia grafica era stata realizzata in Java ed era integrata nel sistema chiamato "Cervello", si è però ritenuto opportuno realizzare anch'essa come componente esterna al sistema. Si è deciso di realizzare l'interfaccia grafica in NodeJs, avvalendosi di Electron. [**?**]

12.2 Electron

Electron è una libreria NodeJs che permette di realizzare delle applicazioni dekstop come se si stesse realizzando un sito web. Questo ha permesso la realizzazione di una semplice applicazione multi piattaforma scrivendo una sola volta il codice.

12.3 Monitoraggio dei canali

L' interfaccia grafica è realizzata come un componente esterno al sistema, i messaggi devono quindi essere inviati agli altri componenti del sistema attraverso i canali di comunicazione. Questo ha quindi permesso una maggior similarità tra l'interazione vocale e quella grafica, consentendo un debug del sistema più efficiente.

12.4 Interfaccia

Nella parte superiore della finestra è presente un campo di input dove viene inserito il comando che vuole essere eseguito. Alla pressione del bottone send verrà inviato attraverso il canale socket.io al "Cervello". Dopo l'invio del comando viene visualizzata nella parte centrale della finestra la risposta data dal sistema. Nel caso in cui la risposta sia positiva viene visualizzata un icona verde. Nel caso non sia stato possibile determinare un comando verrà visualizzata un icona rossa, infine nel caso manchi un parametro viene visualizzata un icona gialla.
Nella parte bassa della finestra infine è presente un campo di testo dove vengono ricevuti i comandi dei qual il cervello ha richiesto l'esecuzione.

Capitolo 13

Elaborazione del comando

Quando il sistema riceve un comando attraverso uno dei canali di input, viene inizializzata la pipeline di elaborazione del comando. Questi sono i principali passaggi :

- Determinazione della distanza da domini
- Determinazione della distanza dalle operazioni
- Ricerca dei parametri
- Assegnazione dei parametri
- Calcolo dei valori di confidence

Nel caso il miglior comando abbia un valore di confidence sufficientemente alto esso viene eseguito. Se non fosse possibile allora vengono avviati i metodi alternativi di determinazione del comando :

- Ricerca in memoria
- Ricerca di parametri per il comando precedente

Capitolo 14

Ricerca Dominio e Operazione

Dopo aver trasformato quanto detto dall'utente in testo, il sistema prova a mappare la frase su una delle azioni che che il sistema è in grado di eseguire. In particolare viene cercato un dominio di esecuzione dell'operazione (es. Meteo, Luci) e l'operazione che si vuole compiere in questo dominio (es. Ricerca condizioni metereologiche, accensione della luce).

L'obbiettivo di questo sistema è di rendere generica l'interazione vocale, si è quindi dovuta trovare una metodologia che identificasse dominio e operazione indipendentemente dalle singole parole utilizzate, cercando quindi di astrarre il significato dai vocaboli.

Il sistema per poter determinare l'operazione dispone di una lista di parole associate all'operazione, una lista di parole associate al dominio e opzionalmente una lista di frasi per l'invocazione dell'operazione in un dominio specifico.

14.1 Database lessicale : wordNet

Il primo approccio si è basato sull'utilizzo di wordNet, un database lessicale della lingua inglese. In wordNet ad ogni lemma è associata una definizione, come in un normale dizionario, ma i lemmi sono anche collegati da una serie di relazioni, formando un grafo. In particolare le relazioni utilizzate nel progetto sono antonimia, iperonimia, sinonimia, metonimia.[**?**]

Utilizzato questo strumento è possibile reperire i sinonimi di una parola, dividendoli per categorie grammaticali (es. sinonimi di light come aggettivo o come nome). Attraverso il confronto delle definizioni e delle relazioni è poi possibile determinare la similarità tra due vocaboli. Per definire tale metrica è possibile utilizzare molti diverse metodologie, tra i quali si è scelto di utilizzare quello definito come *path*. [**?**]

Questo approccio conta il numero di nodi che compongono il percorso più breve tra i due vocaboli, restituisce poi un coefficiente di similarità che corrisponde all'inverso della distanza precedentemente calcolata. Due vocaboli il cui significato è molto simile avranno delle strette relazioni con parole simili, quindi un alto coefficiente di similarità.[**?**]

14.1.1 Calcolo della similarità

La similarità di un dominio con una frase viene calcolata come il massimo della similarità tra tutte le parole associate al dominio con tutte le parole presenti nella frase.
Lo stesso procedimento viene applicato alle operazioni e la similarità della coppia dominio/operazione è costituita dalla media aritmetica delle due similarità.

14.1.2 Vantaggi

Vantaggi:

- **Richiede poche risorse**: il database pesa meno di 100Mb.
- **Supporto ai phrasal verbs**: sono già compresi nel database come un singolo verbo (es. turn_off).

14.1.3 Svantaggi

- **Dizionario statico**: è stato creato manualmente e non viene aggiornato da anni.
- **Indipendente dalla frase**: il risultato dell'analisi di similarità è indipendente dall'ordine delle parole.
- **Dipendente dal pos**: per cercare i sinonimi solo nella corretta categoria grammaticale è necessario affidarsi al Part Of Speech tagger, la cui affidabilità non è totale.

14.2 Part-Of-Speech tagging

L'approccio precedentemente descritto presenta lo svantaggio di non tenere conto dell'ordine delle parole nella frase; si è quindi pensato di aggiungere un componente che tenga in considerazione questa informazione.
Il Part Of Speech tagger è un componente che si occupa di assegnare a ogni parola un tag, si occupa inoltre di definire le relazioni che intercorrono tra questi tag.[**?**][**?**]

14.2.1 Calcolo della similarità

La prima fase consiste nella ricerca di un nome che corrisponde al dominio, da essa vengono poi seguite delle relazioni che legano il verbo a colui che compie l'azione o colui che la subisce. Nei verbi viene quindi cercata l'azione con l'approccio di path similarity attraverso wordnet.

14.2.2 Vantaggi

- **Dipendente dalla frase**: grazie alle relazioni tra le parole viene realmente tenuto conto del senso della frase, migliorando quindi l'affidabilità del sistema.

14.2.3 Svantaggi

- **Eccessivamente dipendente dal POS**: l'analisi si basa completamente sui risultati del POS, sia per le relazioni che per i TAG, nel caso di errori di questo componente (il cui risultati non è sempre affidabile) l'intera analisi è corrotta.

I vari componenti POS (spaCy, ClearNLP, CoreNLP, MATE, Turbo, SyntaxNet) hanno un affidabilità intorno al 93%[**?**], questo tipo di analisi è estremamente difficoltosa nel caso di frasi la cui sintassi non è perfettamente corretta o eccessivamente gergale.
Lo scopo del progetto è di rendere il più possibile flessibile il sistema, questo approccio avrebbe aumentato l'affidabilità, al prezzo di rendere il sistema funzionante solo per frasi con una sintassi perfetta. Abbiamo quindi scelto di togliere questo componente dal sistema.

14.3 Doc2vec

Gli approcci decritti fino ad ora si basano sulla linguistica, nel campo dell'NLP negli ultimi anni sono stati compiuti dei passi avanti nel campo del machine learning. Approcci in questo campo hanno raggiunto performance comparabili, se non migliori, del precedente stato dell'arte.
La maggior parte degli algoritmi di machine learning richiedono la trasformazione del testo in un vettore di dimensione fissa; generalmente l'approccio utilizzato è chiamato bag-of-words , esso però non tiene in considerazione l'ordine delle parole e la loro semantica.[**?**]
Doc2vec è un algoritmo che, come bag of words, permette di rappresentare una parola, una frase, un paragrafo o un intero documento in un vettore di dimensione fissa. Risultati empirici hanno dimostrato che i vettori catturati attraverso doc2vec sono significativi e riescono a condensare l'informazione di un testo di lunghezza variabile.[**?**]

14.3.1 Creazione del modello

L'algoritmo precedentemente descritto per il suo funzionamento necessità di dati precedentemente classificati, se sul disco non è presente nessun modello ne viene creato uno che associa le frasi associate alle operazioni (campo textInvocation del JSON di input) all'operazione stessa.
Se è già presente un modello queste frasi vengono aggiunte a quelle presenti. Dopo aver completato il modello viene avviata la fase di training dei vettori.

14.3.2 Feedback

Il sistema necessità di una grande mole di dati per l'allenamento dei vettori, si è quindi introdotto nel sistema di interazione vocale un metodo di feedback. Dopo l'esecuzione di un comando si ha la possibilità di notificare l'agente se il comando eseguito non è quanto si era richiesto, in caso contrario viene assunto che il comando sia corretto. L'insieme delle frasi corrette viene quindi utilizzato successivamente per l'istruzione dei vettori.

14.3.3 Calcolo della similarità

Dopo aver istruito il modello e aver creato i vettori per ogni frase viene calcolato il vettore medio di tutte le frasi che rappresentano la stessa operazione. Quando l'utente pronuncia una frase viene calcolato l'angolo tra il vettore calcolato per la frase stessa e tutti i vettori medi precedentemente calcolati. Viene poi restituita all'utente la categoria associata al vettore più vicino

$$\textit{similarity(A,B)} = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|}.$$

14.3.4 Vantaggi

- **Dizionario dinamico**: il modello viene creato automaticamente e viene migliorato con il tempo.
- **Dipende dalla frase**: l'algoritmo considera l'ordine delle parole, cattura quindi la semantica della frase.
- **Miglioramento continuo**: il sistema è in grado di apprendere, migliora quindi con il tempo.
- **Indipendente dalle parole**: l'algoritmo apprende, quindi riesce ad imparare nuove parole.

14.3.5 Svantaggi

- **Grande mole di dati**: Per avere una classificazione accurata è necessario avere un grande numero di frasi associate a ogni operazione e ad ogni dominio
- **Piccole variazioni**: L'algoritmo fatica a percepire piccole variazioni che però modificano totalmente il senso della frase (turn on vs. turn off)

14.3.6 Sviluppi futuri

Si è empiricamente dimostrato che in una fase iniziale è difficile creare un modello sufficiente ad istruire il sistema. Non è quindi realizzabile un sistema che si basi solo su questo approccio nei tempi del progetto di bachelor. Si è comunque scelto di realizzare l'intera infrastruttura, cosi che con il passare del tempo il modella possa migliorare fino ad essere in grado di sostituire o almeno collaboratore con l'infrastruttura attuale.

14.4 Word2vec

I modelli precedentemente possono essere racchiusi in due categorie:

- **Statici**: wordnet, pos. Sono basati su un dizionario, non migliorano con il tempo.

- **Dinamici**: doc2vec, skip-thoughts vector. Sono basati su tecniche di deep learning, migliorano con il tempo, ma necessitano di un modello di allenamento.

Si è quindi cercato di trovare un approccio che riuscisse a eliminare la staticità derivante dall'utilizzo di un dizionario, senza però dover creare un modello di allenamento di una rete neurale. Per fare ciò si è ideato sistema basato sull'algoritmo word2vec, il quale, attraverso una rete neurale a due livelli, prende come input del testo e restituisce una serie di vettori.
Questo algoritmo è molto rapido nella sua fase di apprendimento, per questo motivo è possibile allenarlo su una grande base di dati (es. l'intero wikipedia), cosi che i vettori ottenuti siano generici.[**?**]

14.4.1 Calcolo della similarità

Per poter associare una frase pronunciata dall'utente ad un dominio e un operazione viene sfruttata, come in doc2vec, la similarità del coseno. In particolare vengono confrontati i vettori rappresentati le parole associate al dominio (words del JSON) con tutte le parole nella frase, dopo di che viene ritornata la maggior similarità trovata.
Lo stesso procedimento viene applicato alla ricerca dell'operazione.
La similarità finale è calcolata come la media aritmetica della similarità del dominio con quella dell'operazione.

14.4.2 Approccio ad albero

Grazie a word2vec la similarità tra due parole viene ridotta a un prodotto scalare tra due vettori, il che è eseguito molto rapidamente. Il sistema calcola quindi il valore si similarità di tutti i domini, poi calcola la similarità di tutte le operazioni associate a tutti i domini e non solo il migliore. Viene poi scelta l'operazione che possiede la similarità del dominio associato + similarità dell'operazione stessa maggiore.

14.4.3 Vantaggi

- **Dizionario dinamico**: il modello viene creato automaticamente, può essere addestrato su qualunque base di dati.
- **Efficiente**: l'intero calcolo della similarità è ricondotto a prodotti scalari.
- **Indipendente dalle parole**: l'algoritmo apprende, quindi riesce ad imparare nuove parole.

14.4.4 Svantaggi

- **Scelta del modello**: le prestazioni e le risorse utilizzate dipendono dalla scelta del modello. Maggiori informazioni nel capitolo dedicato.
- **Parole composte**: con le classiche tecniche di allenamento i vettori vengono create per singole parole, non sono quindi supportati dal sistema i phrasal verbs.

14.4.5 Parole composte

Nel caso a un'operazione o a un dominio si voglia associate una parola composta, ad esempio "turn off", essa deve essere aggiunta al vettore delle parole associate seguendo la convenzione dei nomi "Lower Camel Case". Il calcolo della similarità verrà poi effettuato come la media della similarità dei token che compongono la parola.[**?**]

14.4.6 Phrasal verbs

Come evidenziato negli svantaggi il sistema non sarebbe in grado di analizzare la similarità una una parola composta, ad esempio "turn on", e una parola singola che possiede lo stesso significato, ad esempio "activate". Per questo motivo, in fase di caricamento del sistema, quando vengono analizzate le parole associate a ogni operazione o dominio, un modulo dedicato si occupa di aggiungere automaticamente sinonimi della parola.
Nella maggior parte dei casi nei sinonimi è contenuto un verbo della categoria opposta (un phrasal verb per un verbo composto da una parola singola e l'inverso). Il problema potrebbe essere risolto con un modello word2vec che contiene phrasal verbs e parole composte.

Capitolo 15

Ricerca dei parametri

Come descritto nei capitoli precedenti, ai comandi posso essere assegnati dei parametri, che appartengono a determinati tipi, definiti da ParameterType.
Per ogni tipologia di parametro si è creata una struttura capace di trovare valori della specifica tipologia di parametro in una frase.

15.1 Assegnazione dei parametri

Il software si occupa di ricercare i parametri di ogni tipologia possibile, dopo aver determinato i valori assunti dai parametri essi vengono assegnati ai comandi.
Inizialmente si era deciso di ricercare i valori dei parametri solo dei comandi che dopo aver determinato dominio e operazione avevano una maggior probabilità di essere eseguiti, successivamente si è scelto di utilizzare l'informazione relativa ai parametri per la determinazione della probabilità di esecuzione di un comando, per questo motivo la ricerca dei valori assunti dai parametri deve essere effettuata per ogni tipologia.

15.1.1 Aumento confidenza comando

Quando viene trovato un parametro appartenente a una determinata tipologia esso viene assegnato a tutti i comandi che possiedono quel tipo di parametro. L'assegnazione provoca un aumento della percentuale che quel comando sia uno dei comandi più coerenti con la frase sentita. Il peso di questo aumento verrà calcolato solo successivamente.

15.2 Color

$$I\ want\ my\ light\ to\ turn\ \underbrace{red} \quad \Longrightarrow \quad Color = [255, 0, 0]$$

$$Set\ light\ color\ to\ \underbrace{dark\ slate\ blue} \quad \Longrightarrow \quad Color = [72, 61, 139]$$

Il sistema è in grado didi ricercare i colori secondo la nomenclatura X11; essa associa a ogni nome di colore i suoi valori nei formati : RGB, HSL/HSV, HSL, HSV. Si è scelto di utilizzare il formato RGB, che viene rappresentato come un vettore composto da 3 elementi di tipo intero.[**?**]
La ricerca del colore nella frase viene effettuata tramite un' espressione regolare, nel caso nella frase compaia un colore composto da più parole, viene scelto il colore con un maggior dettaglio. Per esempio "I want my lamp to turn light blue" restituirà il colore "Light blue" e non "Blue".

15.3 Number

$$Set\ the\ volume\ level\ to\ \underbrace{75.5} \quad \Longrightarrow \quad Number = +75.5$$

Il sistema è in grado di riconoscere numeri, espressi attraverso cifre. Questo è possibile in quanto i sistemi di Speech to Text utilizzati esprimono traducono i numeri espressi attraverso parole in cifre.
I numeri possono rappresentare delle percentuali, avere cifre decimali (usando come separatore '.') e un segno di positività o negatività. La ricerca viene effettuata attraverso un'espressione regolare.

15.4 Location

$$What's\ the\ weather\ like\ on\ the\ \underbrace{Alps} \quad \Longrightarrow \quad Location = Alps$$

$$How'\ s\ the\ weather\ in\ \underbrace{London}\ ? \quad \Longrightarrow \quad Location = London$$

Il sistema è in grado di riconoscere nella frase le parole che rappresentano un luogo. I luoghi vengono rappresentati direttamente attraverso il loro nome, in quanto la maggior parte delle API (es. yahoo meteo) permette di sottomettere richieste direttamente in questo formato.
Il riconoscimento viene effettuato sfruttando il componente di CRF Named Entity Recognizer, parte del pacchetto StanfordNLP. [**?**]

15.4.1 Stanford Named Entity Recognizer

Stanford NER è un implementazione Java di un software in grado di assegnare delle etichette a sequenze di parole, che rappresentano persone, organizzazioni e luoghi. Il software viene offerto insieme a un modello allenato per l'inglese, utilizzato nella nostra implementazione, ma potrebbe essere allenato con altri dati nel caso fosse necessario ad esempio cambiare lingua.

15.5 Date Time

$$Remind\ me\ that\ \underbrace{the\ day\ after\ tomorrow} \quad \Longrightarrow \quad DateTime = 2016{-}08{-}14$$

$$Wake\ me\ up\ \underbrace{at\ 9\ pm,\ tomorrow\ morning} \quad \Longrightarrow \quad DateTime = 2016{-}08{-}20T21:00$$

$$Turn\ on\ the\ light\ \underbrace{every\ 15\ minutes} \quad \Longrightarrow \quad DateTime = T15MX$$

Il sistema è in grado di riconoscere orari, date, espressioni temporali, eventi periodici. La ricerca dei valori di tipo DateTime viene effettuata attraverso il componente SUTime, compreso nella suite StanfordNLP.[**?**]

15.5.1 Stanford SUTime

E' un componente che fa parte della suite di natural language processing di Stanford; esso è basato su un sistema di regole, il suo output è quindi deterministico.
Le regole utilizzate da questo componente sono disponibili solo per la lingua inglese, è quindi uno dei pochi componenti del nostro sistema che impone l'utilizzo della lingua inglese per l'intero sistema.

15.6 Parametri nella frase sucessiva

Nel caso il sistema dovesse riconoscere un comando, per il quale è mancante un parametro, esso è in grado di ricercare il valore assunto del parametro nella frase successiva.

15.6.1 Esempio conversazione con parametro

UTENTE: Viki, Set the light intensity.
SISTEMA: A number is missing!
UTENTE: I would like 80%.
SISTEMA: Done.

Capitolo 16

Determinazione miglior comando

Il valore di confidenza che viene assegnato ad un comando viene assegnato in 3 differenti fasi:

- **Dominio**: Corrisponde alla distanza minima tra le parole associate al domino e quelle presenti nella frase.
- **Operazione**: Corrisponde alla distanza minima tra le parole associate all'operazione e quelle presenti nella frase.
- **Parametri**: Valore positivo nel caso siano stati trovati parametri assegnabili al comando, valori negativi se sono stati trovati parametri non assegnabili al comando.

Dati questi tre parametri è necessario decidere quale comando è più probabile che sia quello indicato nella frase; inoltre è necessario determinare se il comando più probabile deve essere realmente eseguito o se la sua aderenza a quanto pronunciato dall'utente non è sufficiente.

$$\boldsymbol{D_d} = Distanza\ tra\ dominio\ e\ frase\ pronunciata$$

$$\boldsymbol{D_o} = Distanza\ tra\ operazione\ e\ frase\ pronunciata$$

$$\boldsymbol{P_t} = Numero\ di\ parametri\ assegnabili\ al\ comando$$

$$\boldsymbol{C} = Confidenza\ attribuita\ al\ comando$$

$$\boldsymbol{C} = f(D_d, D_o, P_t)$$

16.1 Funzioni statiche

Per poter definire il valore di confidenza da assegnare ad ogni comando si è reso necessario definire una funzione, che tenga in considerazione distanza dal domino, dall'operazione e il contributo dei parametri. Inizialmente si è scelto di definire una funzione che intuitivamente potesse rispettare l'importanza delle varie componenti. La funzione scelta è la seguente :

$$\boldsymbol{C} = f(D_d, D_o, P_t) = 0.5 * D_d + 0.5 * D_o + 0.3 * P_t$$

Essa tiene in considerazione ugualmente il contributo fornito dal domino e quello dell'operazione. Infine garantisce un bonus ai comandi ai quali è stato assegnato un parametro e un malus ai comandi che non possiedono i parametri delle tipologie trovate nella frase.

16.1.1 Vantaggi

- **Veloce**: La funzione è un semplice calcolo, non richiede quindi operazioni computazionalmente complesse.
- **Facilmente gestibile**: Non basandosi su tecniche di machine learning lo sviluppatore può modificare direttamente la funzione.
- **No Data set**: Non richiede l'utilizzo di un dataset per fare il training del sistema.

Per questi motivi questa versione è stata utilizzata durante le fasi di sviluppo del sistema.

16.1.2 Svantaggi

- **Statica**: Non migliora con il passare del tempo.
- **Difficile da generalizzare**: Sarebbe molto complesso riuscire a definire manualmente una funzione che dia un risultato corretto con tutte le combinazioni di parametri.

16.2 Machine learning

L'approccio esposto nella sezione precedente conferisce troppa staticità al sistema, si è quindi scelto di ideare una metodologia di determinazione della confidenza assegnata a un comando attraverso tecniche di machine learning.
Tra i vari approcci disponibili si è scelto di utilizzare naive bayes, per la sua rapidità di calcolo. E' infatti necessario calcolare il valore di confidenza per ognuno dei possibili comandi che potrebbe essere eseguito; il cui numero potenzialmente potrebbe essere molto elevato.

16.2.1 Naive bayes classifier

L'algoritmo di naive bayes viene utilizzato per la classificazione, assumendo che le caratteristiche osservate siano indipendenti tra loro, la classificazione viene effettuata attraverso il teorema di Bayes.

$$P(A \mid B) = \frac{P(B \mid A)\, P(A)}{P(B)}$$

Questo assunzione può essere fatta anche se gli eventi non sono perfettamente indipendenti, è infatti stato dimostrato che le performance sono comunque buone. Questo algoritmo è caratterizzato dalla tua velocità, la probabilità viene infatti calcolata in maniera diretta attraverso delle operazioni aritmetiche.
Grazie a quest'algoritmo è possibile calcolare la possibilità di appartenenza a una classe, date le caratteristiche dell'oggetto.

16.2.2 Discretizzazione

Naive Bayes richiede che le feature siano discrete; per il funzionamento del sistema è inoltre importante che il numero di categorie sia il minore possibile, non è quindi possibile utilizzare direttamente il valore di distanza utilizzato fino ad ora nel sistema, che è salvato il un valore di tipo *double*.
Per ovviare questo problema si è scelto creare 10 categorie di distanza, segmentando a tratti di 0.1 il valore precedentemente calcolato.

16.2.3 Vantaggi

- **Veloce**: Il tempo di calcolo di naive bayes è costante.
- **Dinamico**: Il sistema impara con il tempo, continua a migliorare e riesce a gestire la situazione limite.

Data la dinamicità e le migliori performance del sistema si è scelto di utilizzare questo sistema nell'implementazione finale.

16.2.4 Svantaggi

- **Minor controllo**: Non si può modificare direttamente la funzione di calcolo della confidence.
- **DataSet**: Per il corretto funzionamento del sistema è necessario definire un dataset, che serve durante la fase di training e di verifica.

Capitolo 17

Algoritmo di apprendimento

L'approccio utilizzato per associare i comandi alla frasi pronunciati dall'utente è basato sulla similarità delle parole, ci sono però alcuni casi nei quali alcune espressioni con lo stesso significato utilizzano vocaboli molto differenti tra loro.

Per poter far comprendere queste casistiche al sistema sarebbe stato necessario rivoluzionare l'algoritmo, si è quindi optato per aggiungere una funzione al sistema che permettesse di insegnare comandi arbitrari al sistema.

L'aggiunta di questo algoritmo permette quindi all'utente finale di poterlo programmare, interagendo direttamente con la voce. Questo rende possibile la creazione di comandi rapidi per l'esecuzione di operazioni comuni ed è possibile insegnare al sistema espressioni gergali.

17.1 Comando di apprendimento

Per poter insegnare un comando al sistema è necessario che l'ultima frase sottoposta al sistema sia stata riconosciuta come sconosciuta.

Il riconoscimento del comando di apprendimento viene eseguito esattamente come avviene per gli altri comandi, infatti durante la fase di caricamento dell'universo viene iniettato un comando fittizio che serve per apprendere. Questo garantisce flessibilità al sistema, che anche in questa circostanza non è legato a una frase specifica.

Dopo che il sistema si è dichiarato pronto ad apprendere esso rimane in attesa di una frase dalla quale esso è in grado di determinare un comando, può quindi essere un ricordo già presente o direttamente un comando.

Dopo la conferma di avvenuto apprendimento il sistema sarà in grado di riconoscere il nuovo comando e cercherà di generalizzarlo per riconoscere comandi simili ad esso.

17.2 Generalizzazione del comando

La memoria dei sistema è costituita da una mappa composta di coppie frase, comando. Il sistema però non è solo in grado di riconoscere esattamente la frase presente nel ricordo. Durante la fase di ricerca nella memoria il sistema si occupa di confrontare la frase pronunciata dall'utente con quelle presenti nel ricordo, se uno dei ricordi non è identico a quanto pronunciato, ma è sufficientemente simile, il comando viene restituito.

17.2.1 Similarità di frasi

Per confrontare i ricordi legati alle frasi si è ideato un semplice algoritmo che permette di calcolare la similarità tra frasi. Esso sfrutta lo stesso modello *word2vec* usato nella ricerca di domini e operazioni.
Il coefficiente di similarità è costituito dalla media aritmetica della distanza minima tra le parole che compongono la frase e le parole che compongono il ricordo.

17.2.2 Esempio conversazione

UTENTE: Viki, Goodnight.
SISTEMA: I'm sorry, i do not know what does it mean
UTENTE: Let me teach you something
SISTEMA: Ok, i'm ready to learn.
UTENTE: Turn off the light.
SISTEMA: Learned.

UTENTE: Viki, Goodnight.
SISTEMA: Done.

17.3 Persistenza della memoria

La struttura che rappresenta la memoria è costituita da un oggetto Java, per questo motivo la persistenza è stata realizzata attraverso la serializzazione.
Durante la fase di spegnimento del sistema la memoria viene persistita su file, all'avvio del sistema viene controllato se è presente un file rappresentante la memoria e nel caso sia presente viene utilizzato come memoria attuale.

Capitolo 18

Configurazione

I vari moduli realizzati necessitano diverse configurazioni per poter funzionare correttamente. Si è quindi scelto di esporre tutti i parametri in un file di properites, che viene letto durante l'avvio dei vari sistemi.
Cosí facendo è possibile cambiare gli indirizzi i rete, i valori dei canali di comunicazione e i percorsi per i file senza dover modificare il codice sorgente.
Nel caso alcuni parametri fossero mancanti o considerati errati sono presenti dei valori di default. Le proprietà configurabili sono :

- **WordNet dictionary path**: Percorso del file del database di wordnet.
- **Color dictionary path**: Percorso del file del file di dizionario dei colori.
- **Viki universe path**: Percorso del file da utilizzare come universo, nel caso si voglia caricare da file.
- **Log path**: Percorso del file da utilizzare come log file.
- **Memory path**: Percorso del file da utilizzare come memoria.
- **Viki get URL**: Indirizzo della chiamata che espone la rappresentazione dell'universo.
- **Command receiver address**: Indirizzo al quale avviare il server socket.io
- **Command receiver port**: Port al quale avviare il server socket.io
- **Text command message**: Namespace dei comandi socket.io da ricevere.
- **Viki address** : Indirizzo al quale inviare i comandi da eseguire.
- **Command message** : Namespace dei comandi socket.io da inviare.

- **Min confidence** : Valore di confidenza sotto al quale un comando non è considerato valido.
- **Learning rate**: Valore sopra il quale due frasi vengono considerate equivalenti.

Capitolo 19

Risultati

Capitolo 20

Modelli word2vec

Il sistema descritto nella sezione precedente basa il calcolo della similarità sui vettori estratti dal modello. Essi sono direttamente correlati al corpus [**?**] utilizzato per l'istruzione della rete neurale. Nel web sono presenti modelli già allenati su basi di dati di grandi dimensioni [**?**].
Data la dimensione di del corpus sono presenti un insieme di parole che è molto superiore a quelle che possono essere utilizzate all'interno di un ambiente di automazione casalinga. Sempre grazie alla mole di dati però questi vettori rappresentano pienamente il senso di una parola, al punto di essere utilizzabili in numerosi contesti.

20.1 Google word2vec

Nella prima implementazione del sistema si è scelto di utilizzare il modello fornito insieme all'implementazione di google, il quale è stato addestrato sull'intero google news. I suoi vettori possiedono 300 dimensioni e rappresentano 3 milioni di parole. Questo modello è molto generico, ma le sue dimensioni sono elevate, infatti il sistema necessità di circa 8GB di memoria RAM per il suo funzionamento.

20.2 Allenamento di un Modello

Per migliorare le performance del sistema si potrebbe sostituire il modello fornito da google con un modello addestrato su parole e frasi che realmente possono essere di interesse all'interno di un ambiente domestico. Per questo motivo il lo stesso sistema di logging delle frasi utilizzato da doc2vec viene utilizzato per istruire un modello proprietario. Nei tempi del progetto di bachelor si ritiene che non sia possibile raggiungere la massa di informazioni necessaria a creare

dei vettori che riescano a catturare il significato delle parole. Esso potrebbe essere uno sviluppo futuro del sistema, dopo qualche mese di utilizzo quotidiano. L'implementazione realizzata contiene le funzioni necessarie all'istruzione di un modello ed è predisposta a sostituire il modello attuale.

20.3 sense2vec

I modelli allenati con word2vec associano a ogni parola un vettore, anche quando questa parola assume più significati, il vettore tende poi verso il significato più comune. Per ovviate questo problema è nato sense2vec, il quale associa a ogni parola più vettori, uno per ogni significato. É possibile ad esempio creare il vettore corrispondente a una parola associata al suo Part Of Speech Tag. [**?**] [**?**]
Con lo stesso approccio è anche possibile unire in un vettore più parole, ad esempio i phrasal verbs, cosi da poter mappare un senso e non una sola parola in un vettore. Si è quindi pensato che, una volta raggiunta la necessaria quantità di informazioni, si potrebbe creare un modello seguendo questo approccio, cosi da eliminare il dizionario wordnet dal sistema.

Capitolo 21

Sviluppi futuri

Capitolo 22

Conclusione

Bibliografia

[1] Deep Learning. https://www.technologyreview.com/s/513696/deep-learning/.

[2] Deng Li and Li Xiao. Machine Learning Paradigms for Speech Recognition:An Overview. https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/tasl-deng-2244083-x_2.pdf.

[3] Tomas Mikolov. Distributed Representations of Words and Phrases and their Compositionality. https://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf.

[4] Agente Inteligente. http://www.mind.ilstu.edu/curriculum/ants_nasa/intelligent_agents.php.

[5] Istituto sistemi informativi e networking. http://www.supsi.ch/isin_en.

[6] Snowboy, Hot keyword detection. https://snowboy.kitt.ai.

[7] CMU Sphinx. http://cmusphinx.sourceforge.net.

[8] Snowboy, CPU and RAM usage. https://snowboy.kitt.ai/docs.

[9] Python speech recognizer. https://pypi.python.org/pypi/SpeechRecognition/.

[10] Electron. http://electron.atom.io.

[11] Princeton University "About WordNet." WordNet. Princeton University. 2010. http://wordnet.princeton.edu.

[12] WordNet word similarity methods. http://search.cpan.org/~tpederse/WordNet-Similarity/.

[13] WordNet word path similarity. http://search.cpan.org/~tpederse/WordNet-Similarity/lib/WordNet/Similarity/path.pm.

[14] Part-of-speech tagging. http://www.computational-logic.org/iccl/master/lectures/summer06/nlp/part-of-speech-tagging.pdf.

[15] Part-of-speech categories Treebank. https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html.

[16] Part-of-speech benchmarks. https://spacy.io.

[17] Text Classification from Labeled and Unlabeled Documents using EM. http://www.kamalnigam.com/papers/emcat-mlj99.pdf.

[18] Le Quoc and Mikolov Tomas. Distributed Representations of Sentences and Documents. http://arxiv.org/pdf/1405.4053v2.pdf.

[19] Lower Camel Case. http://c2.com/cgi/wiki?LowerCamelCase.

[20] X11 Color names. https://en.wikipedia.org/wiki/X11_color_names.

[21] Trond Grenager Jenny Rose Finkel and Christopher Manning. 2005. Incorporating Non-local Information into Information Extraction Systems by Gibbs Sampling. Proceedings of the 43nd Annual Meeting of the Association for Computational Linguistics (ACL 2005), pp. 363-370. http://nlp.stanford.edu/~manning/papers/gibbscrf3.pdf.

[22] Trond Grenager Jenny Rose Finkel and Christopher Manning. 2005. Incorporating Non-local Information into Information Extraction Systems by Gibbs Sampling. Proceedings of the 43nd Annual Meeting of the Association for Computational Linguistics (ACL 2005), pp. 363-370. http://nlp.stanford.edu/~manning/papers/gibbscrf3.pdf.

[23] Text corpus. http://language.worldofcomputing.net/linguistics/introduction/what-is-corpus.html.

[24] Word2vec trained models. https://github.com/3Top/word2vec-api#where-to-get-a-pretrained-models.

[25] Trask Andrew, Michalak Phil, and Liu John. sense2vec - A Fast and Accurate Method for Word Sense Disambiguation In Neural Word Embeddings. http://arxiv.org/abs/1511.06388.